Indepedent Study report

Xin Bai

1 Introduction

There are three sources that limit the resolution of CT image. They are the size the light source, the photon diffusion and the pixel size of the detector. We focus on solving the image resolution problem caused by the pixel size of the detector in this report.

According to Nyquist sampling theorem, a continuous signal can be fully reconstructed if it was sampled using the frequency that is twice the highest frequency that signal contains. However, due to the limited size of the pixel, the sampling rate is limited, thus it is unable to capture some high frequency information. And this will lead to the aliasing in the image.

This report implements a dealiasing method which uses a sequence of low-resolution image with small movement to try to estimate a high-resolution image. The relative movement should be smaller than the size of the pixel. With the , we can build a linear relation between the high resolution image and low resolution image. So we can reconstruct the high-resolution image.

2 Method

The idea for reconstruction is that the low resolution image could be obtained by downsampling the high resolution image. To be more explicitly, each low resolution pixel could be viewed as the weighted-averaging result of the corresponding high resolution pixel. The relation could be described as:

$$y = Az + n \tag{1}$$

$$y_i = \Sigma_N w(s)_n z_n \tag{2}$$

, in which z represents the high resolution image, y is the low resolution image and A is the coefficient matrix that specify the contribution pixel for each low-resolution pixel, n is the nosie, i is a pixel in low-resolution image, N is the total number of pixels in y, and w_n denotes the contribution of each high-resolution pixel to y_i and . For y that have relative movement to z, partial contribution should be considered, whose value depends on overlap between moving image and still image. The parameter s in $w(s)$ reflect this situation.

However, due to the errors in mechanics, the real movement does not equal the stage shift as we acquire the data image. We can use MAP method to estimate the best result of high resolution image z and the movement parameters s. So our goal becomes find $\hat{z}$ and $\hat{s}$ that maximizes the $P(z, s|y)$. Next we assume z has a Gaussian prior, the final cost function could be written as :

$$L(z, s) = \frac{1}{2\sigma_n^2}(y - W_s z)^T(y - W_s z) + \frac{1}{2}z^T C_z^{-1} z \tag{3}$$

σ is the variance of the Gaussian noise, C_z is the covariance matrix for z. We need to minimize equation (3) with respect to z and s. One could consider the first term as the linear equation error, and the second term is the image prior error that ensures the smoothness of the estimated image.

An iterative gradient descent procedure is applied in order to optimize the cost function (3). Basically, we take derivative for z and s respectively on equation(3), and therefore get their update equations. The detail of the math, procedures and algorithm flows could be seen in [1].

3 Experiment

First, we acquire a set of low-resolution line-pair images as shown in figure 1. The sequence contains 20 images with 0.1 pixel size movement between adjacency two. The zoomed view of the images in sequence are shown in figure 2. In zoomed view, we can clear see the aliasing in the high frequency part of the line pairs. As the line goes upper, it does not keep straight anymore, instead, some kind of curve pattern is observed.

We can assume that the image sequence only has horizontal movement since when we acquire the image, we only allow the detector to move horizontally without any vertical movement. The stage movement is 0.1 pixel distance per time, however, as mentioned before, the real movement might be differ, we need to estimate it on our own. Since it's only horizontal translation, we can just use NCC or block matching method to estimate the movement first instead of estimating it in the gradient descent procedure. I use a registration toolbox to compute the horizontal translation parameter. The estimation result using registration toolbox is shown in Table 1. Then, simplified iterative MAP estimation algorithm is implemented to find the best high-resolution image given the sequence of raw data and the movement information.

A key difficulty when implementing the algorithm is that one cannot actually build up the coefficient matrix. The matrix size would be the number of high-resolution pixels times the number of all low-resolution pixels in the sequence. This will kill the computation. If we made up a small coefficient matrix on our own, it is easy to find that the matrix is quite sparse, and there are some repeat patterns. So we can use some tricks to code the functional form of the matrix multiplication and transpose multiplication without actually constructing the matrix.

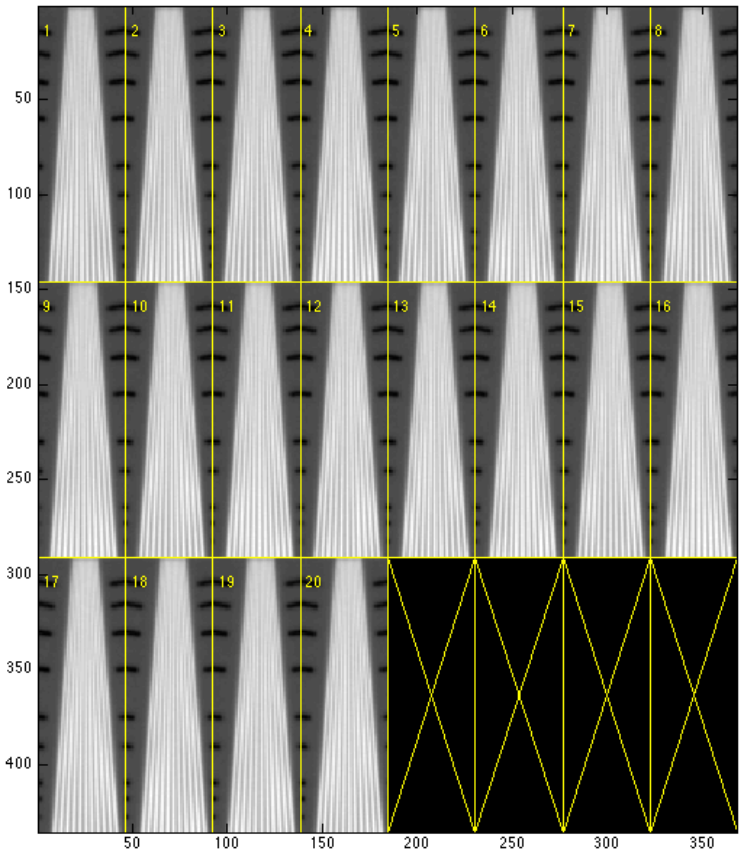

Figure 1: Cropped low-resolution image sequence

To make sure the correctness of the computation and procedure, one can start with a simple toy problem. For example, one can start with trying reconstruct a small single row. In this case, one can actually implement everything over a matrix base, since it's just a single row. It's easy to get things work using matrix form. Once we have that work, we have the ground truth, then we can refer the result of any step in matrix form, which help us locate errors in the functional form.

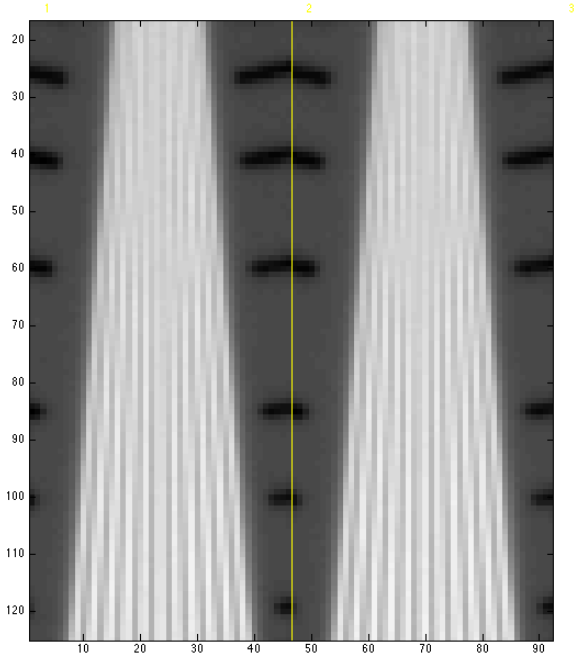

Figure 2: Zoomed Cropped low-resolution image

observe movement (pixel size)	Real movement (pixel size)	Observe movement	Real movement
0.1	0.14	1.0	1.14
0.2	0.18	1.1	1.41
0.3	0.31	1.2	1.48
0.4	0.37	1.3	1.75
0.5	0.53	1.4	1.80
0.6	0.57	1.5	2.04
0.7	0.77	1.6	2.08
0.8	0.82	1.7	2.29
0.9	1.06	1.8	2.34
1.9	2.52	-	-

Table 1: Acquiring movement and Real movement

The result of the iterated MAP estimation is shown in figure 3 . Here, we reconstruct the image with a upsample factor = 2, i.e. each low resolution is

the result of weighted average of two or three(if moved) high resolution pixel. In figure 3, the contrast and structure is much more clear on the upper part of line-pairs.

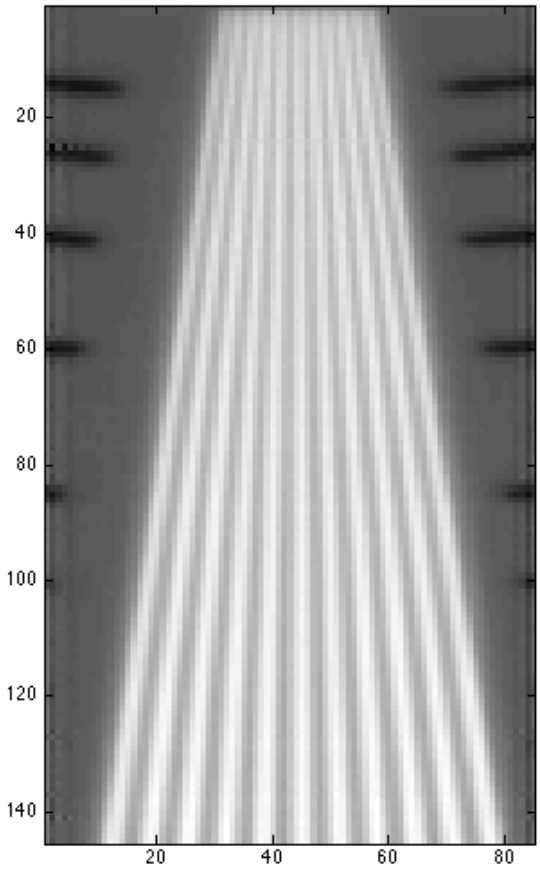

Figure 3: MAP estimation of high resolution image (factor = 2)

4 Discussion

The experiment only reconstruct the high resolution pixel along horizontal direction. It's might not be very difficult for 2D situations if we have already worked out 1D. In order to reconstruct images with 2D movement, we first need a sequence of low resolution images with both horizontal and vertical move-

ments. Then we can use registration method to figure the the translation along both directions. Next, it seems we can do reconstructions by each direction independently.

5 Reference

[1] Hardie, R. C., K. J. Barnard, and E. E. Armstrong. "Joint MAP Registration and High-Resolution Image Estimation Using a Sequence of Undersampled Images." IEEE Transactions on Image Processing 6.12 (1997): 1621–1633. Web. 13 May 2016.

6 Code Appendix

```
% realdata test
clear
close all
clc

%% import image
proj = readImg('linepair_proj.img');
gain = readImg('linepair_gain_proj.img');
proj_off = readImg('linepair_offset_proj.img');
gain_off = readImg('linepair_offset_gain_proj.img');

%% Average
proj_avg = zeros([size(proj,1),size(proj,2),20]);
for i = 1 : size(proj_avg,3)
    proj_avg(:,:,i) = mean(proj(:,:,((i-1)*10 + 1):i*10),3);
end

gain_avg = mean(gain,3);
proj_off_avg = mean(proj_off,3);
gain_off_avg = mean(gain_off,3);

%% get the line pair view and crop it
line_pair = zeros(1000,500,size(proj_avg,3));
for i = 1 : 20
    temp = (proj_avg(:,:,i) - proj_off_avg) ./ (gain_avg - ...
        gain_off_avg);
    %imagesc
    line_pair(:,:,i) = temp;
end

figure; im(line_pair);colormap gray
s_set = ...
    estimate_shift(line_pair(:,:,2:end),squeeze(line_pair(:,:,1)));

%% get the line pair view and crop it
line_pair_crop = zeros(145,46,size(proj_avg,3));
for i = 1 : 20
    temp = (proj_avg(:,:,i) - proj_off_avg) ./ (gain_avg - ...
        gain_off_avg);
    imagesc
    line_pair_crop(:,:,i) = imcrop(temp,[323.5 84.5 45 144]);
end
rows = size(line_pair_crop,1);
cols = size(line_pair_crop,2);
slices = size(line_pair_crop,3);

figure; im(line_pair_crop);colormap gray

%% MAP method, want to reconstruct a high-resolution image.
period =2; %only supersample the pixel at column direction

line_data = line_pair_crop;
```

```
y = ...
    reshape(line_data,[size(line_data,1)*size(line_data,2),size(line_data,3)]);

% initialize z0 by interpolating the first low-resolution image, ...
    only
% interpolate long horizontal direction

x1 = line_data(:,:,1);
x1 = padarray(x1,[0 1],'symmetric','post');
x = (1:size(x1,2));
xv = (1:(1/period):size(x1,2)+((period - 1)/period));
z0 = zeros(size(x1,1),size(x1,2)*period);
for j = 1 : size(x1,1)
    z0(j,:) = interpn(x,x1(j,:),xv,'cubic');
end
%z0(:,1:(period)) = [];
z0(:,end-(period-1):end) = [];
% kernel for the second term in optimization function
kernel = zeros(5);
kernel(1,3) = 1/16;kernel(2,2) = 1/8; kernel(2,3) =-1/2; ...
    kernel(2,4) = 1/8;
kernel(3,1) = 1/16;kernel(3,2) = -1/2;kernel(3,3) = 5/4; ...
    kernel(3,4) = -1/2;kernel(3,5) = 1/16;
kernel(4,2) = 1/8 ;kernel(4,3) = -1/2;kernel(4,4) = 1/8; ...
    kernel(5,3) = 1/16;

%% do reconstruction row by row as each row is individual
for p = 1 : size(line_data,1)
    x1 = z0(p,:);
    y1_move = squeeze(line_data(p,:,:));
    y1_move = y1_move';
    y1_move = y1_move(2:end,:);
    y1_move = padarray(y1_move,[0 1],'both');
    x1 = padarray(x1,[0 2],'both');

    [y1_temp,r_temp] = reconRow(x1,y1_move,1/period,s_set);
    z0(p,:) = y1_temp(3:94);
end
z0 = z0*2;
z0 = z0(:,3:end-5);
figure;imagesc(z0);colormap gray;axis image

%% do reconstruction row by row as an entity
var_eta = mean(mean(var(double(proj_off),0,3)));
lambda = 10;
iter = 100;
hFig = figure();
for i = 1 : iter
    for p = 1 : size(line_data,1)
        x1 = z0(p,:);
        y1_move = squeeze(line_data(p,:,:));
        y1_move = y1_move';
        y1_move = y1_move(2:end,:);
        y1_move = padarray(y1_move,[0 1],'both');
        x1 = padarray(x1,[0 2],'both');

```

```
101          [y1_temp,r_temp] = ...
                 reconRow_real(x1,y1_move,1/period,s_set,var_eta,lambda,kernel);
102          z0(p,:) = y1_temp;
103      end
104      figure(hFig);
105      imagesc(z0);colormap gray; axis off; axis image
106      title(['iter = ', num2str(i)]);
107  end
108
109  %% test sample real,just one row
110
111  close all;
112  clc;
113  row= 3;
114  x1 = z0(row,:);
115
116  y1_move = squeeze(line_data(row,:,:));
117  y1_move = y1_move';
118  y1_move = y1_move(2:end,:);
119  y1_move = padarray(y1_move,[0 1],'both');
120  x1 = padarray(x1,[0 2],'both');
121  figure;imagesc(x1);colormap gray; axis image
122  figure;imagesc(y1_move);colormap gray; axis image
123
124  [y1_temp,r_temp] = reconRow(x1,y1_move,1/period,s_set);
125
126  row= 100;
127  x1 = z0(row,:);
128
129  y1_move = squeeze(line_data(row,:,:));
130  y1_move = y1_move';
131  y1_move = y1_move(2:end,:);
132  figure;imagesc(x1);colormap gray; axis image
133  figure;imagesc(y1_move);colormap gray; axis image
134
135  [y1_temp,r_temp] = reconRow(x1,y1_move,1/period,s_set);
136  %% test toy
137  close all;
138  clc;
139
140  a = y1_temp;
141  imagesc(a);colormap gray;axis image;
142
143  k = 1/2;
144
145  nstep = size(s_set,2);
146  Ws = [];
147  w = 1;
148  downset = zeros(int8(nstep),size(a,2)*k);
149  upset = zeros(int8(nstep),size(a,2));
150  resultconv = zeros(int8(nstep),15);
151  for j = 1:size(s_set,2)
152      % multiplication
153      [downsample,coeff,¬] = matrix_mult(a,s_set(j),k);
154      shift = fix(s_set(j)/k);
155      downsample = downsample/2;
156      downset(w,:) = downsample;
```

```
    w= w+1;
end
downset = downset * 2;
a0 = [zeros(size(a,2)/2,1);ones(size(a,2)/2,1)];
a0 = a0';

a0 = padarray(a0,[0 2],'both');
downset = padarray(downset,[0 1],'both');
figure;imagesc(downset);colormap gray;axis image;
figure;imagesc(a0);colormap gray;axis off;axis image;
[y1_recon,¬] = reconRow(a0,downset*2,k,s_set);
```

```
% do reconstruction on a single row
% ----- a0: high resolution image
% ----- downset: low resolution sequence
% ----- k : downsample factor
% ----- s_set: movement set
% return ---- a0: updated(reconstruted result)
% -----r : residue
function [a0,r] = reconRow(a0,downset,k,s_set)
    %% functional form
    % a sequence of low resolution image with small movement
    y = downset';
    y = y(:);
    iter =1000;
    for i = 1 : iter
        diffset = [];
        gset = zeros(size(a0));
        gamma = [];
        for j = 1 : size(s_set,2)
            [y_temp,¬] = matrix_mult(a0,s_set(j),k);
            y_m = downset(j,:);
            diff_temp = y_temp - y_m;
            diffset = [diffset;diff_temp];
            g_temp = matrix_T_mult(diff_temp,k,s_set(j));
            %gset = [gset;sum(g_temp)];
            gset = gset + g_temp;
        end
        gset = gset*k; % g is ready
        for p = 1 : size(s_set,2)
            [gamma_temp,¬] = matrix_mult(gset,s_set(p),k);
            gamma = [gamma;gamma_temp]; % gamma is ready
        end
        diffset = diffset';
        diffset = diffset(:);
        gamma = gamma';
        gamma = gamma(:);
        epsilon = (gamma'*diffset)/(gamma'*gamma);
        r = norm(epsilon*gset,2)/norm(a0,2);
        a0 = a0 - epsilon * gset;
        if r < 10^(-12)
            break
        end
    end
end
```

```
% This function estimate shift using dft registration algorithm, ...
    the output
% contain the info of horizontal and vertical translation.
% ----- dataset : low resolution sequence except the first ...
    image(moving image)
% ----- ref : the reference image ( the first image at sequence)
function s_set = estimate_shift(dataset,ref)
    s_set = size(dataset,3);
    g = ref;
    for i = 1:size(dataset,3)
        f = squeeze(dataset(:,:,i));
        [output,¬] = dftregistration(fft2(f),fft2(g),100);
        s = output(4);
        s_set(i) = s;
    end
end
```

```
%% function form of matrix multiplication
% ----- a : high resolution image
% ----- Δ: movement
% ----- k : inverse of upsample factor (if upsample factor = 3 , ...
    k = 1/3)

function [downsample,coeff,result_conv] = matrix_mult(a,Δ,k)
    part = mod(abs(Δ),k)/k;
    shift = fix(Δ/k);
    coeff = ones(1,(1/k)+1);

    % build coefficient matrix
    if(Δ ≥ 0)
        coeff(1) = 1 - part;
        coeff(end) = part;
    else
        coeff(1) = part;
        coeff(end) = 1 - part;
    end
    Δ;
    coeff;
    result_conv = conv(a,coeff);
    if Δ ≥ 0
        x = 1/k - fix(mod(Δ,1)/k);
        y = size(a,2)-shift;
    else
        x = 1/k - fix(mod(Δ,1)/k)+1;
        y = size(result_conv,2)-(1/k)+shift+1;
    end

    while x < 0
        x = x +1/k;
    end
    while y > size(result_conv,2)
        y = y - 1/k;
    end
    downsample = result_conv(:,uint8(x):1/k:uint8(y));

```

```
    % padding for keep right size
    if Δ ≥ 1
        downsample = padarray(downsample,[0 ...
            abs(fix(Δ))],'symmetric','pre');
    else
        downsample = padarray(downsample,[0 ...
            abs(fix(Δ))],'symmetric','post');
    end

    %downset(w,:) = downsample;
end
```

```
% the function that performs transpose matrix multiplication ...
    given downsample factor
% and movement
% ----- downsample: low resolution image
% ----- k : downsample factor
% ----- Δ: movement
function temp_upsamp = matrix_T_mult(downsample,k,Δ)
    part = mod(abs(Δ),k)/k;
    coeff = ones(1,(1/k)+1);
    shift = fix(Δ/k);
    if(Δ ≤ 0)
        coeff(1) = 1 - part;
        coeff(end) = part;
    else
        coeff(1) = part;
        coeff(end) = 1 - part;
    end

    dsamp_supp = downsample;

    shift = abs(shift);

    if Δ ≥ 0
        x = 1/k;
    else
        x =1;
    end
    upsample = repmat(dsamp_supp,[1/k,1]);
    upsample = upsample(:);
    y = size(upsample,1);
    if Δ > 0
        for i = x:1/k:y
            if i ≠ y
                upsample(i) = ...
                    coeff(1)*upsample(i+1)+coeff(end)*upsample(i);
            elseif i==y
                if coeff(end) ≠ 0
                    upsample(i) = coeff(end)*upsample(i);
                end
            end
        end
    end

    if Δ < 0
```

```
43          for i = x:1/k:y
44              if i ≠ y
45                  upsample(i) = coeff(1)*upsample(i) + ...
                        coeff(end)*upsample(i+1);
46              elseif i == y
47                  if coeff(1) ≠ 0
48                      upsample(i) = coeff(1)*upsample(i);
49                  end
50              end
51          end
52      end
53
54      % pad to keep size
55      if Δ > 0
56          temp_upsamp = padarray(upsample,[shift 0],'post');
57          temp_upsamp = temp_upsamp(1+shift:end);
58      else
59          temp_upsamp = padarray(upsample,[shift 0],'pre');
60          temp_upsamp = temp_upsamp(1:size(upsample,1));
61  %         temp_upsamp(1:end-shift) = upsample(1:end-shift);
62      end
63      %pause;
64      %figure(hFig3);
65      %imagesc(upsample');colormap gray;axis image;
66      temp_upsamp = k*temp_upsamp';
67  end
```